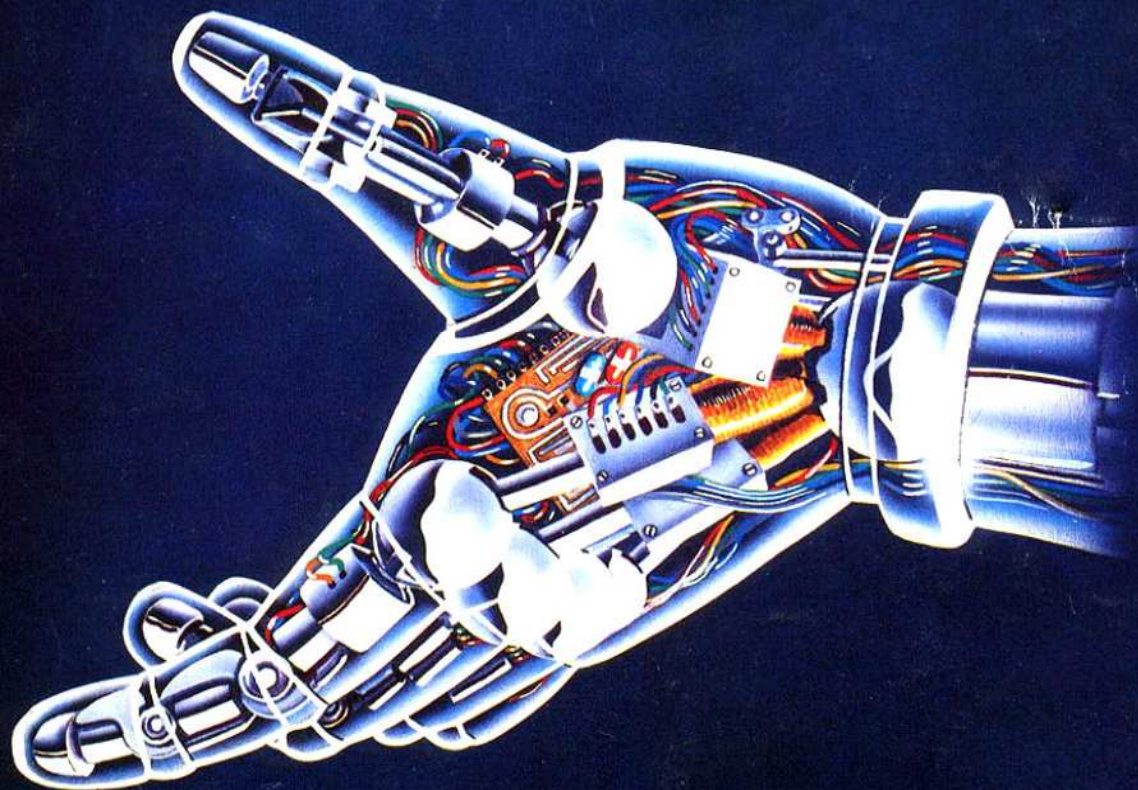
MICRO & PERSONAL computer

51 lire 4.000

m&p COMPUTER - marzo 1985 - n. 51 - Anno VI - mensile - Sped. abb. post. gr. III 70%

PROVE: COMPAQ PLUS,
YASHICA YC 64 contro SPECTRAVIDEO 728 CALCRESULT

SOMMARIO

Grafica: Diana Santosuosso

e danno in uscita il numero t, lunghezza relativa del segmento nella direzione libera trovata; è necessario in tutti e due i casi decrementare di 1 la variabile t. Si tenga presente, infatti, che FOR n=1 TO 100: NEXT n: PRINT n scriverebbe 101.

IL SECONDO PROGRAMMA

Una volta che si faccia funzionare il primo programma, si nota un particolare, tanto più rilevante quanto più estese sono le figure da riempire: durante l'elaborazione, il cammino esegue spesso delle spirali involutorie, e, una volta terminate, rimane a lungo fermo. Alla luce della conoscenza del funzionamento del programma questo evento è chiaro, e rappresenta una duplice perdita di tempo: prima nel disegnare la spirale, poi, di più ancora, nel percorrere a ritroso la spirale nella ricerca di direzioni libere. Un'idea per risolvere questo inconveniente è la seguente: quando il programma si «accorge» di disegnare una spirale, allora ne riempie direttamente e semplicemente l'interno, evitando così di percorrerla sia in senso diretto che in senso inverso. Questo avviene nel secondo programma, che, anzi, fa anche qualcosa in più: il programma cerca, per quanto possibile, di individuare il maggior numero possibile di queste spirali; si è dovuto, a questo proposito, infrangere l'asserto che i nodi figli di uno stesso padre siano ordinati a priori, per decidere volta per volta l'ordinamento che più conviene allo svolgimento del programma.
Bisogna, a questo punto, osservare che il secondo programma riempie correttamente solo figure semplicemente connesse (=senza buchi); infatti, se una spirale avvolge un buco, il secondo programma lo riempie, mentre il primo lo lascerebbe intatto. Spesso però, una figura con buchi può essere facilmente ricondotta a una semplicemente connessa disegnando opportunamente qualche segmento: per esempio la *fig. 3a)* non è semplicemente connessa, la *fig. 3b)* invece lo è.
Non starò qui a dare una descrizione minuziosa anche del secondo programma, che comunque il lettore potrà interpretare alla luce di quanto detto. C'è da dire invece che le modifiche accennate ne appesantiscono notevolmente la struttura e quindi ne rallentano la velocità, per cui risulta veramente più utile solamente quando si tratti di riempire figure abbastanza grandi e di grande area rispetto al perimetro. Nel caso di figure piccole (come quella dell'esempio) l'elaborazione è addirittura più lenta, mentre per riempire tutto lo schermo il tempo è di 7-8 minuti contro un'ora e un quarto del primo programma.

ALCUNE NOTE

Per chi avesse problemi di occupazione di memoria (in genere i possessori del 16k), la matrice a (200,2) causa qualche problema: infatti da sola occupa 2k di memoria. La soluzione proposta, che è quella del listato 3, consiste nel trasformare la matrice a.. da numerica a matrice di caratteri; per far questo però è necessario cambiare il funzionamento del puntatore da relativo ad assoluto. Il programma proposto occupa così circa 1500 bytes, di cui solo 500 di variabili.
Chi possedesse un altro computer, per quanto riguarda la conversione del programma tenga presente le seguenti note sulla grafica dello Spectrum:
lo schermo è costituito da 256x176 pixel, che vanno da 0 a 175 per le ordinate; così a questi numeri devono essere sostituiti quelli relativi al computer in proprio possesso; attenzione alle linee 610 e 710: si legga, ad esempio
610 FOR t= −1 TO 0-y STEP -1 e lo zero deve essere sostituito dal numero opportuno;
l'istruzione PLOT x,y disegna il pixel di coordinate x,y;
l'istruzione DRAW a,b disegna, a partire dall'ultimo punto disegnato (siano x e y le sue coordinate) un segmento di retta fino al punto di coordinate k+a,y+b;
la funzione booleana;
POINT (x,y) dà 1 (=condizione vera) se il pixel di coordinate k,y è acceso (cioè disegnato), 0 (=condizione falsa) altrimenti.

Chi avesse proprio problemi di velocità, oltre a riscrivere il programma in Assembly, può soltanto riscriverlo parzialmente, quanto basta perché sia accettato da un compilatore. I compilatori sono spesso utili (personalmente uso MCODER II), a patto però di rinunciare ad alcune comodità: in genere usano solo piccoli numeri interi, non hanno matrici pluridimensionali, eccetera; ma, a meno che non si abbia proprio la necessità di lavorare con i numeri in virgola mobile, questi problemi si possono facilmente superare. Il difetto principale è che, in genere, per eseguire un programma compilato occorre che il compilatore stesso sia in memoria.

Corrado Zanella

TI99/4A

CAMBIO DELLA GUARDIA SUL FRONTE TEXAS

Dopo aver condotto per oltre un anno e mezzo la rubrica sul Software Texas Instruments TI99/4A — rubrica che ho personalmente visto nascere e crescere in spazio e contenuti accompagnandola fin dal numero 34 di m&p — non sarei certamente sincero se, abbandonandola, dicessi di non avere rimpianti.
Da quel «lontano» giugno del 1983, quando con il precedente Direttore Di Pisa ero entrato ufficialmente in collaborazione con m&p, ad oggi, fredda giornata di gennaio di questo nuovo 1985, le cose sono cambiate parecchio, e non solo per l'avvento del dinamico Boss Gualtiero Rudella: allora doveva ancora esplodere la moda (o mania) del personal computer, il fenomeno che durante quell'inverno avrebbe condizionato l'intero mercato informatico italiano, dando luogo ad una vera e propria «caccia al personal» (ve ne ricordate?), e le riviste specializzate in questo campo si potevano contare sulle dita di una mano. Attualmente le edicole traboccano di fascicoli, nastri magnetici, libercoli e materiale vario che a stento è possibile distinguere: è quello che può essere drasticamente riassunto e definito come «sciacallaggio editoriale». Il discorso vale soprattutto per il Software: vendere programmi per mezzo delle edicole non sarà forse qualificante, ma certamente redditizio; e visto che i computer, adesso, li vende anche il droghiere, perché non farlo?
Attenzione però: una cosa è vendere software (con la esse minuscola), un'al-

tra è fare una Rivista (con la erre maiuscola). m&p (e qualcun altro ad onor del vero) in edicola non si mescola con la folla di neonati giornalini informatici, proprio come l'olio non può essere mischiato con l'acqua.
Nello spazio che m&p mi ha riservato, ho sempre voluto sottolineare il mio, il nostro impegno, considerando solo marginalmente la pubblicazione di banali listati da copiare, e cercando piuttosto di coinvolgere personalmente ogni singolo lettore, affrontando argomenti oscuri ancora da scoprire editorialmente nel parlare del buon vecchio TI99.
Devo dire con orgoglio che la cosa ha funzionato, centrando in pieno il bersaglio prefissato: tantissimi lettori appassionati hanno testimoniato la loro presenza e manifestato il loro pieno appoggio all'iniziativa inviando lettere, notizie, programmi, consigli.

Tirando le somme, un successo di cui — meritatamente! — si poteva andar fieri.

Maggiore il numero dei lettori, maggiore l'impegno con cui, puntualmente, occorre consegnare il materiale dattiloscritto alla Redazione: mensilmente, anche se a qualcuno potrà sembrare esagerato, è un onere non indifferente. Ho iniziato la mia collaborazione che ancora non ero maggiorenne: se il Liceo di allora poteva permettere tempo libero a sufficienza, l'Università di adesso neppure lo concepisce.

Per questo motivo l'abbandono della rubrica è risultato essere, senza drammi, indispensabile.

Cedo la mano all'amico e già collaboratore Paolo Bagnaresi (Dott.), che tutti hanno avuto modo di conoscere ed apprezzare in qualità di vincitore del concorso di aprile, e sulla cui validità non credo sia necessario aggiungere altro.
Chi vorrà, potrà comunque continuare a seguirmi sulle pagine di m&p, attraverso un nuovo spazio dedicato al potente sistema MSX: una comparsa per ora solo bimestrale, che ha reso possibile un compromesso tra studio e...Rudella!
Dell'MSX mi occupo ormai da quasi sei mesi, ed ho avuto modo di conoscerlo approfonditamente grazie all'esperienza di lavoro maturata in una casa di Software. Per molti aspetti l'ho trovato parente stretto del TI99, del quale possiede lo stesso microprocessore video...
Anche se ormai parecchio in ritardo, auguro buon anno a tutti.

DISASSEMBLER

Desidero sottoporvi un interessante programma che sarà estremamente utile per tutti coloro che si occupano di linguaggio macchina e/o di ASSEMBLER: si tratta di un disassembler in BASIC. Un utilissimo tool per ricreare un listato in sorgente, e quindi comprensibile, da un programma in formato oggetto.

CHE COSA È UN DISASSEMBLATORE?

Per capire che cosa vuol dire DISASSEMBLARE, occorre comprendere prima il significato della parola ASSEMBLARE. Un microprocessore, per eseguire un programma, necessita di istruzioni. Queste sono poste in memoria (RAM o ROM non fa differenza, a questo riguardo) come una successione di «CODICI», che sono poi dei particolari valori numerici, che, se ben congegnati, permettono l'esecuzione del programma stesso. Questi «CODICI», variabili da un microprocessore ad un altro, potrebbero essere immessi in memoria direttamente da parte del programmatore. Naturalmente, ciò comporterebbe la perfetta conoscenza del linguaggio macchina del microprocessore impiegato e cioè la capacità di calcolare manualmente tutte le possibili combinazioni che determinano il «codice» finale in linguaggio macchina. È un sistema tediosissimo, sconsigliabile a chi non ha avuto una condanna all'ergastolo, e, oggi, per fortuna, non più usato. Notare, comunque, che agli albori dell'informatica questo era l'unico mezzo disponibile. Un netto miglioramento è stato introdotto con l'uso degli ASSEMBLATORI. Un assemblatore permette l'utilizzo di un comodo sistema MNEMONICO (Linguaggio simbolico, anche se molti discettano se l'assembler sia o non sia un linguaggio). Un assemblatore permette al programmatore l'uso di un linguaggio a sé congeniale, provvedendo, nel contempo, alla pedante trasformazione del linguaggio Mnemonico (Simbolico) usato nelle necessarie istruzioni (codici) in linguaggio macchina. Per esempio un'istruzione in codice macchina esadecimale del nostro microprocessore: >0460, che ha lo stesso significato del GOTO del Basic, è espressa nell'assembler della Texas Instr. (Codice Mnemonico) con una «B» (ove «B» significa «BRANCH»). Altro esempio: >C000 (la LET del Basic) diventa «MOV». È ovvio che questo è un sistema molto più agevole e più facile da ricordare della memorizzazione di tutta quella sequela di numeri che costituiscono un programma in linguaggio macchina. Quindi, nel nostro esempio, il programmatore scriverà B e il microprocessore intenderà >0460. Chiaro?

Tutto l'insieme di istruzioni Mnemoniche (Simboliche) viene in gergo chiamato «SORGENTE». Le istruzioni in codice (linguaggio macchina) che vengono generate partendo dalla «SORGENTE», per mezzo dell'assemblatore, vengono chiamate «OGGETTO». L'assemblatore provvede quindi a trasformare una «SORGENTE» in un «OGGETTO».
Il microprocessore può ESEGUIRE (il RUN del Basic) solo un programma «OGGETTO», ma non la sua «SORGENTE». Ora, è possibile che si possieda l'Oggetto di un programma assembler, ma non la sua Sorgente. Per capire un po' di più quel programma occorrerebbe invece proprio la sua Sorgente. A questo provvede il DISASSEMBLATORE che m&p vi offre questo mese.
Supponiamo, per esempio, di ricevere da un amico un bellissimo quanto velocissimo programma in linguaggio macchina, bello da vedersi, bello da usarsi, ma, per i più, totalmente incomprensibile.
Bene, basterà far girare questo programma per vedere, non molto velocemente, in verità, disassemblato il programma misterioso. Non molto velocemente, dicevo, poiché la velocità (in BASIC) non è certo il cavallo di battaglia del buon vecchio TI 99/4A. In compenso, sarà possibile utilizzare il DISASSEMBLER in BASIC con MINI-MEMORY, in BASIC con modulo EDITOR/ASSEMBLER e in EXT-BASIC.
Infatti il nocciolo del programma sta in una (famigerata) CALL PEEK (LOCX, MX, NX) in riga 4860.
Purtroppo le istruzioni PEEK & POKE sono assenti nel TI-BASIC e solo parzialmente supplite da istruzioni analoghe in MINI-MEMORY o in BASIC ESTESO.

MEMORIA E DISASSEMBLER

Senza espansione di memoria a 48 K, il DISASSEMBLER vi sarà utile solo per disassemblare il contenuto delle ROM da >0000 a >1FFF (ROM della console), e delle ROM della MINI-MEMORY e dell'EXTENDED BASIC, da >6000 a >7FFF (il Modulo EDITOR/ASSEMBLER non ha ROM, ma solo GROM). Non è molto, ma comunque interessante e didattico per l'apprendimento di ciò che è contenuto

nei recessi della memoria della nostra macchina.
Se invece possedete anche l'espansione di memoria, potrete usufruire in toto delle potenzialità del disassemblatore con il Modulo MINI-MEMORY e con il Modulo EDITOR/ASSEMBLER, mentre avrete qualche limitazione con il modulo EXTENDED BASIC. Infatti l'EXTENDED BASIC, con la espansione di memoria inserita, carica sia i programmi BASIC, sia i programmi in linguaggio macchina, in memoria CPU (cioè nell'espansione): se il programma in linguaggio macchina ha una fine di caricamento in alta me-

```
100 REM ********************
110 REM *  DISASSEMBLATORE *
120 REM ********************
130 REM *     by M.KROLL   *
140 REM * da  99' magazine *
150 REM *    marzo  1983   *
160 REM *------------------*
170 REM * adattamento  per *
180 REM * EX-basic,minimem *
190 REM * e gestione files *
200 REM * su dischi a cura *
210 REM *     di R.FABIANI  *
220 REM *  novembre 1984   *
230 REM ********************
240 CALL CLEAR
250 CALL SCREEN(6)
260 FOR I=1 TO 14
270 CALL COLOR(I,16,6)
280 NEXT I
290 DIM S(15)
300 GOSUB 3560
310 PER$="*****************
**********"
320 CALL CLEAR
330 PRINT PER$:"        DISASS
EMBLATORE":PER$
340 PRINT : : : : : : : : :
350 IF PRT$<>"S" THEN 380
360 CLOSE #1
370 PRT$=""
380 IF FIL$="S" THEN 700
390 INPUT "uscita su stampan
te?(S/N) ":PRT$
400 IF PRT$<>"S" THEN 480
410 F=1
420 IF DEVICE$<>"" THEN 450
430 PRINT : :PER$:"stampante
 settata a :": :
440 INPUT "":DEVICE$
450 OPEN #1:DEVICE$
460 CALL CLEAR
470 GOTO 560
480 F=0
490 PRINT : : :
500 INPUT "crea file su disc
o? (S/N) ":FIL$
510 IF FIL$<>"S" THEN 560
520 F=1
530 PRINT
540 INPUT "scegli [DSK1(2,3)
.NOMEFILE]
            ":DIS$
550 OPEN #1:DIS$
560 CALL CLEAR
570 PRINT : :"PREMI 1-disass
embla OPCODE": :"PREMI 2-dis
assembla DATA": :"PREMI 3-di
sassembla TEXT": :"PREMI 4-e
dita file": :"PREMI 5-fine"
580 CALL KEY(0,K,ST)
590 IF ST=0 THEN 580
600 IF (K<49)+(K>53)=-1 THEN
 580
610 CALL CLEAR
620 IF K=52 THEN 4880
630 IF K=53 THEN 5240
640 GOSUB 780
650 IF K=49 THEN 1010
660 IF (K=50)+(F=1)=-2 THEN
4060
670 IF (K=50)+(F=0)=-2 THEN
4430
680 IF (K=51)+(F=1)=-2 THEN
4260
690 IF (K=51)+(F=0)=-2 THEN
4630 ELSE 4880
700 PRINT : :
710 PRINT PER$: :"Vuoi chiud
ere il FILE ": :DIS$: :"(S/N
) ":
720 INPUT CHI$
730 IF CHI$<>"S" THEN 570
740 CLOSE #1
750 F=0
760 FIL$=""
770 GOTO 320
780 REM     INPUT INDIRIZZI D
A DISASSEMBLARE
790 CALL CLEAR
800 INPUT "disassemblare da
?            (indirizzo hex di
 4 cifre)   ":A$
810 PRINT
820 IF POS("13579BDF",SEG$(A
$,LEN(A$),1),1)=0 THEN 840
830 A$=SEG$(A$,1,LEN(A$)-1)&
SEG$("02468ACE",POS("13579BD
F",SEG$(A$,LEN(A$),1),1),1)
840 IF LEN(A$)=4 THEN 870
850 PRINT :"":"l'input deve
essere di 4    cifre in esad
ecimale": :
860 GOTO 800
870 INPUT "fino a ?
             (indirizzo hex di
 4 cifre)   ":B$
880 IF POS("13579BDF",SEG$(B
$,LEN(B$),1),1)=0 THEN 900
890 B$=SEG$(B$,1,LEN(B$)-1)&
SEG$("02468ACE",POS("13579BD
F",SEG$(B$,LEN(B$),1),1),1)
900 IF LEN(B$)=4 THEN 930
910 PRINT : :"l'input deve e
ssere di 4    cifre in esade
cimale": :
920 GOTO 870
930 TEMP$=A$
940 GOSUB 3310
950 A=DEC
960 TEMP$=B$
970 GOSUB 3310
980 B=DEC
990 CALL CLEAR
1000 RETURN
1010 REM     VALORI PEEK & CON
VERSIONE
1020 FOR LOC=A TO B STEP 2
1030 L=0
1040 V1=LOC
1050 GOSUB 3930
1060 LOC$=HEX$
1070 GOSUB 4820
1080 M=MX
1090 N=NX
1100 V=M*256+N
1110 V1=V
1120 GOSUB 3930
1130 V$=HEX$
1140 VA=V
1150 GOSUB 3480
1160 REM     DETERMINA FORMATO
 ISTRUZIONE
1170 IF V<512 THEN 4040
1180 IF V<832 THEN 2830
1190 IF V<1024 THEN 2770
1200 IF V<2048 THEN 2380
1210 IF V<4096 THEN 2250
1220 IF V<8192 THEN 1740
1230 IF V<11264 THEN 1980
1240 IF V<12288 THEN 3140
1250 IF V<14336 THEN 2110
1260 IF V<16384 THEN 3140 EL
SE 1570
1270 REM    STAMPO OPCODE MNE
MONICO
1280 GAP=POS(OPER$," ",1)
1290 IF GAP<6 THEN 1320
1300 E$=OPER$
1310 GOTO 1330
1320 E$=SEG$(OPER$,1,GAP)&SE
G$("        ",1,5-GAP)&SEG$(OP
ER$,GAP+1,LEN(OPER$))
1330 IF F=0 THEN 1360
1340 PRINT #F:LOC$;"     ";V$;
"     ";E$
1350 GOTO 1370
1360 PRINT #F:LOC$;" ";V$;"
";E$
1370 IF L=3 THEN 1400
1380 IF L=2 THEN 1500
1390 IF L<>1 THEN 1540
1400 IF F=0 THEN 1430
1410 PRINT #F:LO$(1);"     ";V
V$(1)
1420 GOTO 1440
1430 PRINT #F:LO$(1);" ";VV$
(1)
1440 IF L=1 THEN 1540
1450 IF F=0 THEN 1480
1460 PRINT #F:LO$(3);"     ";V
V$(3);"     ";OPE$
1470 GOTO 1540
1480 PRINT #F:LO$(3);" ";VV$
(3);" ";OPE$
1490 GOTO 1540
1500 IF F=0 THEN 1530
1510 PRINT #F:LO$(1);"     ";V
V$(1):LO$(2);"     ";VV$(2)
1520 GOTO 1540
1530 PRINT #F:LO$(1);" ";VV$
(1):LO$(2);" ";VV$(2)
1540 NEXT LOC
```

```
1550 GOSUB 5180
1560 GOTO 320
1570 REM   FORMATO I
1580 RESTORE 1720
1590 GOSUB 3890
1600 T$=SEG$(BIN$,11,2)
1610 R$=SEG$(BIN$,13,4)
1620 GOSUB 3420
1630 GOSUB 3630
1640 S$=R$
1650 T$=SEG$(BIN$,5,2)
1660 R$=SEG$(BIN$,7,4)
1670 GOSUB 3420
1680 GOSUB 3630
1690 D$=R$
1700 OPER$=OP$&" "&S$&","&D$
1710 GOTO 1280
1720 DATA 61440,SOCB,57344,S
OC,53248,MOVB,49152,MOV,4505
6,AB,40960,A
1730 DATA 36864,CB,32768,C,2
8672,SB,24576,S,20480,SZCB,1
6384,SZC
1740 REM   FORMATO II
1750 RESTORE 1960
1760 GOSUB 3890
1770 DISP$=SEG$(BIN$,9,8)
1780 DIS=0
1790 FOR X=8 TO 15
1800 DIS=DIS+VAL(SEG$(DISP$,
X-7,1))*S(X)
1810 NEXT X
1820 IF DIS<128 THEN 1840
1830 DIS=DIS-256
1840 IF SEG$(OP$,2,1)="B" TH
EN 1930
1850 IF DIS=0 THEN 1910
1860 DIS=DIS*2+LOC+2
1870 V1=DIS
1880 GOSUB 3930
1890 OPER$=OP$&" >"&HEX$
1900 GOTO 1280
1910 OPER$="NOP"
1920 GOTO 1280
1930 REM   ISTRUZIONI DI CON
TROLLO
1940 OPER$=OP$&" "&STR$(DIS)
1950 GOTO 1280
1960 DATA 7936,TB,7680,SBZ,7
424,SBO,7168,JOP,6912,JH,665
6,JL,6400,JNO,6144,JOC,5888,
JNC,5632,JNE,5376,JGT
1970 DATA 5120,JHE,4864,JEQ,
4608,JLE,4352,JLT,4096,JMP
1980 REM   FORMATO III
1990 RESTORE 2100
2000 GOSUB 3890
2010 T$=SEG$(BIN$,11,2)
2020 R$=SEG$(BIN$,13,4)
2030 GOSUB 3420
2040 GOSUB 3630
2050 S$=R$
2060 R$=SEG$(BIN$,7,4)
2070 GOSUB 3420
2080 OPER$=OP$&" "&S$&","&ST
R$(R)
2090 GOTO 1280
2100 DATA 10240,XOR,9216,CZC
,8192,COC
2110 REM   FORMATO IV
2120 RESTORE 2240
2130 GOSUB 3890
2140 R$=SEG$(BIN$,7,4)
2150 GOSUB 3420
2160 C$=STR$(R)
2170 R$=SEG$(BIN$,13,4)
2180 T$=SEG$(BIN$,11,2)
2190 GOSUB 3420
2200 GOSUB 3630
2210 S$=R$
2220 OPER$=OP$&" "&S$&","&C$
2230 GOTO 1280
2240 DATA 13312,STCR,12288,L
DCR
2250 REM   FORMATO V
2260 RESTORE 2370
2270 GOSUB 3890
2280 R$=SEG$(BIN$,13,4)
2290 GOSUB 3420
2300 S$="R"&STR$(R)
2310 C$=SEG$(BIN$,9,4)
2320 R$=C$
2330 GOSUB 3420
2340 D$=STR$(R)
2350 OPER$=OP$&" "&S$&","&D$
2360 GOTO 1280
2370 DATA 2816,SRC,2560,SLA,
2304,SRL,2048,SRA
2380 REM   FORMATO VI
2390 RESTORE 2500
2400 GOSUB 3890
2410 R$=SEG$(BIN$,13,4)
2420 T$=SEG$(BIN$,11,2)
2430 GOSUB 3420
2440 GOSUB 3630
2450 IF R$<"@>6018" THEN 248
0
2460 IF R$>"@>6050" THEN 248
0
2470 GOSUB 2520
2480 OPER$=OP$&" "&R$
2490 GOTO 1280
2500 DATA 1856,ABS,1792,SETO
,1728,SWPB,1664,BL,1600,DECT
,1536,DEC,1472,INCT,1408,INC
2510 DATA 1344,INV,1280,NEG,
1216,CLR,1152,X,1088,B,1024,
BLWP
2520 REM   PROGRAMMI UTILITY
 DI MINI-MEMORY
2530 DATA 6018,GPLLNK,601C,X
MLLNK,6020,KSCAN,6024,VSBW,6
028,VMBW,602C,VSBR,6030,VMBR
2540 DATA 6034,VWTR,6038,DSR
LNK,603C,LOADER,6040,NUMASG,
6044,NUMREF,6048,STRASG,604C
,STRREF,6050,ERR
2550 RESTORE 2530
2560 READ U$,UTIL$
2570 IF SEG$(R$,3,4)<>U$ THE
N 2560
2580 IF F=0 THEN 2600
2590 R$=R$&"  ("&UTIL$&")"
2600 IF U$="6018" THEN 2640
2610 IF U$="6038" THEN 2640
2620 IF U$="601C" THEN 2640
2630 RETURN
2640 L=3
2650 LOC=LOC+2
2660 V1=LOC
2670 GOSUB 3930
2680 LO$(3)=HEX$
2690 GOSUB 4820
2700 M2=MX
2710 N2=NX
2720 V1=256*M2+N2
2730 GOSUB 3930
2740 VV$(3)=HEX$
2750 OPE$="DATA >"&HEX$
2760 RETURN
2770 REM   FORMATO VII
2780 RESTORE 2820
2790 GOSUB 3890
2800 OPER$=OP$
2810 GOTO 1280
2820 DATA 992,LREX,969,SKOF,
928,SKON,896,RTWP,864,RSET,8
32,IDLE
2830 REM   FORMATO VIII
2840 RESTORE 3130
2850 GOSUB 3890
2860 R$=SEG$(BIN$,13,4)
2870 GOSUB 3420
2880 D$="R"&STR$(R)
2890 LOC=LOC+2
2900 L=L+1
2910 V1=LOC
2920 GOSUB 3930
2930 LO$(L)=HEX$
2940 GOSUB 4820
2950 M1=MX
2960 N1=NX
2970 V1=256*M1+N1
2980 GOSUB 3930
2990 VV$(L)=HEX$
3000 S$=">"&HEX$
3010 IF OP$="LIMI" THEN 3070
3020 IF OP$="LWPI" THEN 3070
3030 IF OP$="STST" THEN 3090
3040 IF OP$="STWP" THEN 3090
3050 OPER$=OP$&" "&D$&","&S$
3060 GOTO 1280
3070 OPER$=OP$&" "&S$
3080 GOTO 1280
3090 LOC=LOC-2
3100 L=L-1
3110 OPER$=OP$&" "&D$
3120 GOTO 1280
3130 DATA 768,LIMI,736,LWPI,
704,STST,672,STWP,640,CI,608
,ORI,576,ANDI,544,AI,512,LI
3140 REM   FORMATO IX
3150 RESTORE 3300
3160 GOSUB 3890
3170 R$=SEG$(BIN$,13,4)
3180 T$=SEG$(BIN$,11,2)
3190 GOSUB 3420
3200 GOSUB 3630
3210 S$=R$
3220 R$=SEG$(BIN$,7,4)
3230 GOSUB 3420
3240 IF OP$<>"XOP" THEN 3270
3250 D$=STR$(R)
3260 GOTO 3280
3270 D$="R"&STR$(R)
3280 OPER$=OP$&" "&S$&","&D$
3290 GOTO 1280
3300 DATA 15360,DIV,14336,MP
Y,11264,XOP
3310 REM   CONVERTE IN DECIM
ALE
3320 DEC=0
3330 FOR X=3 TO 15 STEP 4
3340 TEMP2$=SEG$(TEMP$,(X+1)
/4,1)
3350 IF ASC(TEMP2$)>57 THEN
```

```
3400
3360 TN=ASC(TEMP2$)-48
3370 DEC=DEC+TN*S(X)
3380 NEXT X
3390 RETURN
3400 TN=ASC(TEMP2$)-55
3410 GOTO 3370
3420 REM   PONE REGISTRO #
3430 R=0
3440 FOR X=12 TO 15
3450 R=R+VAL(SEG$(R$,X-11,1)
)*S(X)
3460 NEXT X
3470 RETURN
3480 REM   CONVERTE A BINARI
O
3490 BIN$=""
3500 FOR X=0 TO 15
3510 BIN=INT(VA/S(X))
3520 VA=VA-(BIN*S(X))
3530 BIN$=BIN$&STR$(BIN)
3540 NEXT X
3550 RETURN
3560 REM   CREA DIVISORE BIN
ARIO
3570 DATA 32768,16384,8192,4
096,2048,1024,512,256,128,64
,32,16,8,4,2,1
3580 RESTORE 3570
3590 FOR X=0 TO 15
3600 READ S(X)
3610 NEXT X
3620 RETURN
3630 REM   IDENTIFICA CAMPO-
T
3640 IF T$<>"00" THEN 3670
3650 R$="R"&STR$(R)
3660 RETURN
3670 IF T$<>"01" THEN 3700
3680 R$="*R"&STR$(R)
3690 RETURN
3700 IF T$<>"11" THEN 3730
3710 R$="*R"&STR$(R)&"+"
3720 RETURN
3730 LOC=LOC+2
3740 L=L+1
3750 GOSUB 4820
3760 M1=MX
3770 N1=NX
3780 V1=LOC
3790 GOSUB 3930
3800 LO$(L)=HEX$
3810 V1=M1*256+N1
3820 GOSUB 3930
3830 VV$(L)=HEX$
3840 IF R<>0 THEN 3870
3850 R$="@>"&HEX$
3860 RETURN
3870 R$="@>"&HEX$&"(R"&STR$(
R)&")"
3880 RETURN
3890 REM   CREA SIMBOLO MNEM
ONICO DELL'OPCODE
3900 READ OPV,OP$
3910 IF V<OPV THEN 3900
3920 RETURN
3930 REM   CONVERTE IN ESADE
CIMALE
3940 HEX$=""
3950 FOR X=3 TO 15 STEP 4
3960 VH=INT(V1/S(X))
3970 V1=V1-VH*S(X)
3980 IF VH>9 THEN 4020
3990 HEX$=HEX$&STR$(VH)
4000 NEXT X
4010 RETURN
4020 HEX$=HEX$&CHR$(VH+55)
4030 GOTO 4000
4040 OPER$="CODICE OGGETTO N
ON VALIDO"
4050 GOTO 1280
4060 REM   PRINT <DATA> SU D
EVICE ESTERNO
4070 FOR LOOP=A TO B STEP 18
4080 V1=LOOP
4090 GOSUB 3930
4100 L$=HEX$
4110 PRINT #F:L$:"
 ":"DATA ";
4120 FOR LOC=LOOP TO LOOP+16
 STEP 2
4130 GOSUB 4820
4140 M=MX
4150 N=NX
4160 V1=256*M+N
4170 GOSUB 3930
4180 IF LOC=LOOP+16 THEN 421
0
4190 IF LOC>=B-1 THEN 4240
4200 PRINT #F:">";HEX$;",";
4210 NEXT LOC
4220 PRINT #F:">";HEX$
4230 NEXT LOOP
4240 PRINT #F:">";HEX$
4250 GOTO 5180
4260 REM   PRINT <TEXT> SU D
EVICE ESTERNO
4270 FOR LOOP=A TO B STEP 54
4280 V1=LOOP
4290 GOSUB 3930
4300 PRINT #F:HEX$;"
  ":"TEXT '";
4310 FOR LOC=LOOP TO LOOP+53
4320 GOSUB 4820
4330 M=MX
4340 IF (M<127)+(M>31)=-2 TH
EN 4360
4350 M=63
4360 PRINT #F:CHR$(M);
4370 IF LOC=B THEN 4410
4380 NEXT LOC
4390 PRINT #F:"'"
4400 NEXT LOOP
4410 PRINT #F:"'"
4420 GOTO 5180
4430 REM   VISUALIZZA <DATA
> SU SCHERMO
4440 FOR LOOP=A TO B STEP 6
4450 V1=LOOP
4460 GOSUB 3930
4470 L$=HEX$
4480 PRINT #F:L$;" DATA ";
4490 FOR LOC=LOOP TO LOOP+4
STEP 2
4500 GOSUB 4820
4510 M=MX
4520 N=NX
4530 V1=256*M+N
4540 GOSUB 3930
4550 IF LOC=LOOP+4 THEN 4580
4560 IF LOC>=B-1 THEN 4610
4570 PRINT #F:">";HEX$;",";
4580 NEXT LOC
4590 PRINT #F:">";HEX$
4600 NEXT LOOP
4610 PRINT #F:">";HEX$
4620 GOTO 5180
4630 REM   VISUALIZZA <TEXT>
 SU SCHERMO
4640 FOR LOOP=A TO B STEP 14
4650 V1=LOOP
4660 GOSUB 3930
4670 PRINT #F:HEX$;" TEXT '"
;
4680 FOR LOC=LOOP TO LOOP+13
4690 GOSUB 4820
4700 M=MX
4710 IF (M<127)+(M>31)=-2 TH
EN 4730
4720 M=63
4730 PRINT #F:CHR$(M);
4740 IF LOC=B THEN 4780
4750 NEXT LOC
4760 PRINT #F:"'"
4770 NEXT LOOP
4780 PRINT #F:"'"
4790 GOSUB 5180
4800 GOTO 320
4810 REM   PEEK ROUTINE
4820 IF LOC<32768 THEN 4850
4830 LOCX=LOC-65536
4840 GOTO 4860
4850 LOCX=LOC
4860 CALL PEEK(LOCX,MX,NX)
4870 RETURN
4880 REM INPUT FILE
4890 PRINT PER$: :"File da d
isco:DSK(123).NOME": :
4900 INPUT "":DIS$
4910 CALL CLEAR
4920 OPEN #3:DIS$,INPUT
4930 IF EOF(3)THEN 5130
4940 CALL KEY(0,KY,ST)
4950 IF ST=0 THEN 5000
4960 FOR KY=1 TO 300
4970 NEXT KY
4980 CALL KEY(0,KY,ST)
4990 IF ST=0 THEN 4980
5000 IF ZX$="" THEN 5040
5010 EDI$=ZX$
5020 ZX$=""
5030 GOTO 5060
5040 INPUT #3:EDI$,
5050 IF EOF(3)THEN 5150
5060 INPUT #3:ZX$,
5070 IF POS(ZX$," ",5)=5 THE
N 5110
5080 PRINT EDI$&","&ZX$
5090 ZX$=""
5100 GOTO 4930
5110 PRINT EDI$
5120 GOTO 4930
5130 PRINT ZX$
5140 GOTO 5160
5150 PRINT EDI$
5160 CLOSE #3
5170 ZX$=""
5180 PRINT : :PER$:"**** FIN
E DISASSEMBLATO ****"
5190 PRINT PER$:TAB(9):"Prem
i un tasto":PER$
5200 CALL SOUND(150,1397,0)
5210 CALL KEY(0,KY,ST)
5220 IF ST=0 THEN 5210
5230 GOTO 320
5240 END
```

moria sopra > D5CD, decimale -10803, (inizio programma DISASSEMBLER in alta memoria), esso distruggerà il nostro programma BASIC preesistente, cioè annienterà DISASSEMBLER. Con il Modulo MINI-MEMORY o con il Modulo EDITOR/ASSEMBLER questo rischio non esiste: il programma BASIC sta sempre in VDP RAM, e non interferirà mai con il programma in linguaggio macchina (necessarie, minimo, 16 ore di tempo!).
Se caricate un file oggetto che non contiene AORG (Absolute ORiGin = inizio di caricamento), esso si piazzerà: da >24F4 in sù con l'EXTENDED BASIC, da >A000 in sù con il modulo MINI-MEMORY o EDITOR/ASSEMBLER. Gli eventuali files oggetto successivi, sempre senza AORG, saranno caricati di seguito al primo.
Infine, nella MINI-MEMORY priva di espansione di memoria, i programmi in linguaggio macchina, caricati da cassetta, saranno situati nella piccola RAM della Mini Memory stessa e cioè, generalmente, da >7D00 in sù. In taluni casi, se sono stati rilocati, possono iniziare anche da >7118 in sù.

COME USARE IL DISASSEMBLER

Dovrete caricare per primo il DISASSEMBLER e poi il programma oggetto, da disco mediante una CALL LOAD («DSKn.[NOMEFILE]»), da cassetta con un programma apposito che vi forniremo insieme al programma XASSEMBLER, un assemblatore in BASIC, di prossima pubblicazione.
Date il RUN e seguite pedissequamente le istruzioni che appariranno sul monitor.

ENTRIAMO NEL DISASSEMBLER

È obbligatorio premettere che il DISASSEMBLER non è stato programmato dal sottoscritto (come spiegato nelle prime REM) ma solo modificato e ottimizzato per un uso più ampliato riguardo ai propositi dell'autore.
Dopo aver eseguito tutto quanto spiegato fino ad ora, diamo il RUN. Lo schermo cambiarà colore, titolo, prima richiesta: «USCITA SU STAMPANTE? (S/N)». Rispondete S o N. Se S(i) vi verrà chiesto come è stata settata.
Digitate i valori che di solito usate per la vostra «OPEN».
Comparirà poi la richiesta «CREA FILE SU DISCO? (S/N)». Se volete registrare il disassemblato direttamente su disco rispondete S(i).
In questo caso dovrete digitare il tipo di unità dischi in questione (n = 1,2 o 3) e il nome scelto per il file. (Es = «DSK (n.).Nomefile»)
Lo schermo si pulisce ed appare il MENÙ principale:
PREMI 1-disassembla OPCODE
PREMI 2-disassembla DATA
PREMI 3-disassembla TEXT
PREMI 4-edita file
PREMI 5-fine
Apparirà ora la richiesta di indirizzo di partenza.
Digitatela e apparirà la richiesta dell'in-

ALCUNE APPLICAZIONI

Due esempi del disassemblato ottenibile con il DISASSEMBLER sono a fianco di questo articolo:

— Il primo va da locazione > 0000 a locaz. > 007A: questa parte di memoria contiene la routine di inizializzazione che si ha con il RESET (FCTN QUIT).

— Il secondo, molto breve, da locaz. > 0A6A a locaz. > 0A82, contiene la speciale CALL KEY, effettuata dall'interprete Basic, preposta al controllo del tasto FCTN QUIT. Durante il RUN di un programma Basic questa routine viene eseguita in continuazione: se premete FCTN QUIT avrete, come noto, l'immediato RESET del sistema operativo che si esplica mediante la BLWP ο > 0000 che si trova alla locaz. > 0A80.
Questi due esempi si riferiscono a routines presenti nella memoria ROM della consolle, quindi essi sono ottenibili indipendentemente dal tipo di modulo impiegato per lanciare il DISASSEMBLER.
Si consiglia di effettuare il disassemblato di queste due zone di memoria, e di confrontare il disassemblato ottenuto da Voi con i due listati da noi forniti. Se essi coincidono, ci sono 90% di probabilità che non abbiate commesso errori di battitura, altrimenti... riguardatevi il listato del programma con più attenzione: avete commesso qualche errore.
Cercate di impratichirvi con questo DISASSEMBLER. Basta possedere l'EXTENDED BASIC, la consolle, un registratore a cassette e naturalmente... aver battuto con estrema cura il DISASSEMBLER.

Riccardo Fabiani

```
0000   83E0   C    @>0024.R15
0002   0024
0004   83C0   C    R0.R15
0006   0900   SRL  R0.0
0008   83C0   C    R0.R15
000A   0A92   SLA  R2.9
000C   30AA   LDCR @>0460(R10).2
000E   0460
0010   02B2   STWP R2
0012   0008   CODICE OGGETTO NON VALIDO
0014   1E00   SBZ  0
0016   0460   B    @>007A
0018   007A
001A   1E00   SBZ  0
001C   0460   B    @>0078
001E   0078
0020   0460   B    @>04B2
0022   04B2
0024   020D   LI   R13.>9800
0026   9800
0028   020E   LI   R14.>0100
002A   0100
002C   020F   LI   R15.>8C02
002E   8C02
0030   0200   LI   R0.>0020
0032   0020
0034   1013   JMP  >005C
0036   1000        NOP
0038   1E00   SBZ  0
003A   02E0   LWPI >280A
003C   280A
003E   0380        RTWP
0040   280A   XOR  R10.0
0042   0C1C   SRC  R12.1
0044   FFD8   SOCB *R8.*R15+
0046   FFF8   SOCB *R8+.*R15+
0048   83A0   C    @>8300.R14
004A   8300
004C   1100        NOP
004E   06A0   BL   @>0864
0050   0864
0052   06A0   BL   @>0864
0054   0864
0056   C90D   MOV  R13.@>8300(R4)
0058   8300
005A   C342   MOV  R2.R13
005C   D11D   MOVB *R13.R4
005E   C180   MOV  R0.R6
0060   DB46   MOVB R6.@>0402(R13)
0062   0402
0064   DB60   MOVB @>83ED.@>0402(R13)
0066   83ED
0068   0402
006A   5820   SZCB @>011B.@>837(
006C   011B
006E   837C
0070   0300   LIMI >0002
0072   0002
0074   0300   LIMI >0000
0076   0000
0078   D25D   MOVB *R13.R9
007A   1105   JLT  >0086
```

```
0A6A   020C   LI   R12.>0024
0A6C   0024
0A6E   30E0   LDCR @>0012.3
0A70   0012
0A72   0B7C   SRC  R12.7
0A74   020C   LI   R12.>0006
0A76   0006
0A78   3605   STCR R5.8
0A7A   2560   CZC  @>004C.5
0A7C   004C
0A7E   1602   JNE  >0A84
0A80   0420   BLWP @>0000
0A82   0000
```

dirizzo di fine disassemblando. Fornite il valore richiesto.
Ora abbiate pazienza, c'è chi lavora per voi!
Passato un po' di tempo (sigh!), il disassemblato, su schermo, su carta, o su disco, sarà completato.
Ritornerete così al MENÚ principale.

ALCUNE CONSIDERAZIONI

Il disassemblatore può disassemblare indifferentemente ogni locazione di memoria in tre maniere: OPCOTE, DATA, TEXT. Con OPCODE il programma partirà dall'assunto che i contenuti delle locazioni in esame siano effettivi CODici OPerativi, cioè istruzioni in L.M. Tenterà quindi di darvi il codice mnemonico corrispondente. Tuttavia, può accadere che le locazioni in questione contengano o dei DATA o delle stringhe (TEXT). Imperterrito, il nostro disassemblatore interpreterà sia i DATA che le stringhe come istruzioni in L.M.: si avrà quindi un disassemblato incoerente, nel quale sembrerà che il programmatore era, al momento della stesura del programma, ubriaco o peggio.

Quando ciò accade, sappiate che certamente quella particolare zona di memoria conteneva o TEXT o DATA. Provate quindi a disassemblare con l'opzione TEXT.

Se ancora non otterrete risultati plausibili resta solo un'ultima soluzione: l'opzione DATA. Infatti questa è un'opzione che fornisce sempre valori coerenti.
Lasciare sempre questa opzione per ultima, perché non vi sarà poi di grande aiuto tutta quella sfilza di valori esadecimali: occorre trovare quale zona del programma faceva uso di quei DATA, prima di aver vantaggio dal disassemblato ottenuto. Da notare, inoltre, che nel disassemblato ottenuto per OPCODE, i DATA sono sempre presenti: li trovate subito alla destra del valore esadecimale della locazione.

Disassemblare con l'opzione DATA è utile solo per avere un listato più compatto.

Un'ultima osservazione: sporadicamente, sempre con l'opzione OPCODE, può capitare di trovare, per una sola locazione di memoria o comunque per poche locazioni (in genere fino a 3-5 locazioni), istruzioni o strane o una segnalazione di «CODICE OGGETTO NON VALIDO». Fate caso: se questi valori seguono delle BL <Branch and Link>, le GOSUB del BASIC, o delle BLWP <Branch and Load Workspace Pointer>, le CALL del BASIC, allora quasi certamente essi sono dei DATA che devono essere passati alle sobroutine chiamate (tipico è la BLWP @@DSRLNK, DATA 8: il disassemblatore darà «CODICE OGGETTO NON VALIDO», per il DATA 8, perché 8 non è un codice possibile per nessuna istruzione in L.M. del TMS 9900).

Se esse invece seguono dei JMP <jump> o dei B <Branch> (il GOTO del BASIC), esse sono invece semplici DATA, messi lì da un programmatore svogliato, che non ha avuto pazienza di riunirli tutti insieme in un unica tavola dei DATA.

Chi non ha ancora sufficienti conoscenze di assembler, si limiti a provare prima con OPCODE e poi con TEXT: riuscirà, dopo un'attenta analisi di ciò che ottiene, a capire ugualmente qualcosa ..... perlomeno dove sono situate le stringhe (TEXT)!

All'inizio dovrete fare mente locale a ciò che desiderate ottenere dal programma, ma con un po' di esperienza ne sarete soddisfatti.

ANALISI DEL LISTATO

100-770 settaggio colori e MENÚ
780-1000 input indirizzi da disassemblare
1010-1150 valori PEEK e conversione
1160-1260 determina formato istruzione
1270-1560 stampa OPCODE mnemonico
1570-1730 formato I
1740-1920 formato II
1930-1970 istruzioni di controllo
1980-2100 formato III
2110-2240 formato IV
2250-2370 formato V
2380-2510 formato VI
2520-2760 programmi utility MINI-MEMORY
2770-2820 formato VII
2830-3130 formato VIII
3140-3300 formato IX
3310-3410 converte in decimale
3420-3470 pone registro (numero)
3480-3550 converte in binario
3560-3620 crea divisore binario
3630-3880 crea campo T
3890-3920 crea simbolo mnemonico dell'OPCODE
3930-4050 converte in esadecimale
4060-4250 visualizza DATA
4260-4420 visualizza TEXT
4430-4620 visualizza DATA su monitor
4630-4800 visualizza TEXT su monitor
4810-4870 PEEK routine
4880-5170 edita FILE su monitor

Cercate di impratichirvi con questo DISASSEMBLER. Basta possedere l'EXTENDED BASIC, la consolle, un registratore a cassette e naturalmente .... aver battuto con estrema cura il DISASSEMBLER.

Riccardo Fabiani

TI 99/4A

XASSEMBLE: ASSEMBLER PER EXTENDED BASIC

Come promesso dal nostro ottimo predecessore, Paolo Ventafridda (passato a curare il nuovo computer della serie MSX), per la parte assembler della rubrica del TI-99/4A iniziamo la pubblicazione della prima parte di quello che diventerà un assembler completo per Extended Basic.
Il programma, una volta ultimato, sarà utilizzabile da coloro che possiedono il Modulo Extended Basic, registratore a cassetta, ed espansione di memoria 48 K, tipo ESSEMMECI, per intenderci. Tut-

```
*** SPOLIN e' uno SPOstatore di riferimento LINee di un programma EXTENDED BASIC.
* Dopo la CALL LINK("SPOLIN") da Extended tutte le linee del programma Extended
* eventualmente presente in memoria vengono fatte puntare, e quindi coincidono,
* con le istruzioni che erano presenti nella linea di programma piu' bassa.
*
* SPOLIN crea all'interno di un programma Ext. basic lo spazio necessario per
* contenere successivamente un programma in Linguaggio Macchina.
*
* SPOLIN e' da usarsi per preparare XASSEMBLE. In particolare per preparare
* LINEA25000, il nucleo nel quale verra' prima inserito CSAVE, e sotto il quale
```

to il sistema è infatti stato studiato tenendo in mente che molti hanno l'espansione di memoria del tipo «laterale», me non il sistema a dischi.
Questo ha complicato notevolmente la realizzazione di un valido sistema di salvataggio (SAVE) del codice in linguaggio macchina (L.M.) che viene generato dall'assemblatore.
L'assemblatore è made in Germany. Credo che l'autore sia Karl Hagenbuchner, di Traun, ma non ci giurerei, visto che conosco solo poche parole di tedesco.
È merito dell'inesauribile Ventafridda la scoperta di questo validissimo programma. L'originale tedesco conteneva una serie di GOTO errate, che impedivano l'uso corretto di tutte le JUMP (salti condizionati e non) ad un LABEL (etichetta).
È stata da me aggiunta la routine di SAVE e LOAD (chiamata d'ora in poi «CSALO») del codice macchina generato.

STRUTTURA DI XASSEMBLE

XASSEMBLE è composto di due tronconi: XASSEMBLER vero e proprio, che è un normale programma Ext. Basic, e CSALO, che è un programma in L.M., ottenuto da Ext. Basic tramite CALL LOAD, e successivamente rilocato all'interno di un programma BASIC appositamente precostituito.
Molti di voi non sono forse avvezzi a questo tipo di struttura.
È comunque abbastanza usata per degli ottimi motivi: il programma in L.M. che è contenuto in un programma BASIC si può salvare su cassetta, oltre che su disco, è molto veloce da caricare, occupa un decimo dell'area di programma BASIC dell'equivalente che venga generato con CALL LOAD, fà risparmiare spazio sul dischetto e su cassetta, è immediatamente disponibile dopo il «RUN». Questo mese vi descriverò CSALO e i programmi che lo generano. Il prossimo mese sarà la volta, finalmente, di XASSEMBLE.

CSALO

Questo spezzone permetterà il «SAVE» e due tipi diversi di «LOAD», sempre da cassetta. È molto facile la trasformazione della routine per il sistema a dischi, per coloro che possiedono l'EDITOR/ASSEMBLER.
CSALO è la porzione di XASSEMBLE che và creata per prima. Successivamente sarà aggiunto XASSEMBLE.
Chi ha il sistema a Dischi può aggiungere XASSEMBLE mediante l'opzione MERGE. È, in questo, XASSEMBLE che deve essere aggiunto a CSALO e non viceversa.

```
* verra' poi costruito XASSEMBLE, ma funziona per qualsiasi programma Extended
* Basic.
*****************************************************************************
* RILOC e' il rilocatore di CSAVEXT, programma Extended Basic con CALL LOAD.
* CSAVEXT ha il suo ambiente naturale a partire da >FD88, per >0230 bytes, ma e'
* caricato, tramite CALL LOAD, da >24F4 a >2724, per non interferire con il
* programma Extended Basic che deve essere creato in alta memoria.
* RILOC, se chiamato con CALL LINK("RILOC"), rilochera' il tutto cio' che si
* trova in bassa memoria a partire da >24F4 a >2724, e lo spostera' da >FD88
* fino a >FFB8, ove deve trovare LINEA25000 gia' trattato con SPOLIN.
* Variando cio' che si carica (LI R..) in R0, R1, R2, si puo' usare RILOC per
* rilocare qualsiasi altro programma in linguaggio macchina.
*****************************************************************************
* PAOLO BAGNARESI, San Donato Milanese, 9 Dicembre 1984. Tel. (02)-514.202
*****************************************************************************

       DEF  SPOLIN,RILOC
       AORG >2726
SPOWK  BSS  >20            -> 32 bytes per il workspace.
SPOLIN LWPI SPOWK          -> Carica il workspace.
       MOV  @>8330,R0      -> @>8330 c'e' il puntatore all'inizio della tavola
*                             delle linee del programma Ext. Basic in uso.
       MOV  @>8332,R4      -> @>8332 c'e' il puntatore alla fine della tavola
*                             delle linee di un programma Ext. Basic. La tavola
*                             occupa 2 words (4 bytes) per ogni linea di programma
*                             Il primo word e' il numero di linea, il secondo word
*                             e' l'indirizzo CPU RAM dove si trovano le istruzioni
*                             contenute in questo numero di linea.
       DEC  R4             -> Aggiusta in modo di puntare all'indirizzo-istruzioni.
       MOV  R4,R1          -> Fa' una copia in R1.
       MOVB *R1+,R2        -> Ottiene in R2 (MSB) il byte sinistro dell'indirizzo.
       MOVB *R1,R3         -> Ottiene in R3 (MSB) il byte destro dell'indirizzo.
       INCT R0             -> R0 punta ora alla locazione che contiene l'indirizzo
*                             delle istruzioni della linea piu' alta del programma
*                             Extended Basic.
LOOP   MOVB R2,*R0+        -> Sostituisce il byte sinistro con l'indirizzo-istruz.
*                             della linea piu' bassa.
       MOVB R3,*R0+        -> Fa' la stessa cosa con il byte destro.
       INCT R0             -> Si posiziona sul prossimo indirizzo-istruzioni.
       C    R0,R4          -> Vede se e' arrivato alla riga piu' bassa.
       JNE  LOOP
RETURN LWPI >83E0          -> Si allaccia all'interprete GPL e ritorna
       SB   @>837C,@>837C     all'Extended Basic.
       B    @>0070
RILOC  LWPI SPOWK          -> Carica il workspace in uso.
       LI   R0,>24F4       -> In R0 il primo indirizzo utile della bassa memoria,
*                             dove si trova l'inizio del programma da rilocare.
       LI   R1,>FD88       -> In R1 il primo indirizzo di destinazione, dove il
*                             programma sara' rilocato (alta memoria).
       LI   R2,>0230       -> Sono >0230 bytes da rilocare (2*256+3*16=512+48=560)
TRASF  MOV  *R0+,*R1+      -> Riloca (trasferisce) da bassa memoria in alta mem.
       DECT R2             -> Vede se ha finito il trasferimento.
       JNE  TRASF          -> Se R2 non e' ancora zero, continua a trasferire.
       JMP  RETURN         -> Altrimenti ritorna all'Extended Basic.
       END
```

```
*                    SAVE UTILITY PER CASSETTA                    *
*                      IN TI EXTENDED BASIC                       *
*                 DA USARSI INSIEME A XASSEMBLE                   *
*                        PAOLO BAGNARESI                          *
*                      Tel. (02)-514.202                          *
*                       8 Dicembre 1984                           *
*******************************************************************
* FORMATO DEL PROGRAM CHE VERRA' GENERATO IN VDP RAM
*       VDP
*       RAM
* Word  Loc         Contenuto della locaz.
*
*  1  >1000     --> Identifier NEXT PROGRAM (>0000 o >FFFF)
*  2  >1002     --> Quanti Bytes in questo segmento di program
*  3  >1004     --> Inizio di caricamento di questo segmento
*  4  >1006     --> Primo word del programma da salvare.
*  .   ....    :
*  .   ....    : --> PROGRAM BUFFER AREA.
*  .   ....    :
*  .   ....    :
*  X  >XXXX     --> Ultimo word del programma da salvare.
* X+1 >XXXX+2   --> Lunghezza DEF/REF table (bytes)
* X+2 >XXXX+4   --> Prima lettera della DEF piu' bassa.
*...  ....     :
*...  ....     :
*...  ....     : --> DEFs effettive  (8 bytes per ciascuna DEF)
*...  ....     :
*X+N/2 XXXX+N   --> Settimo e ottavo byte dell'ultima DEF, la piu' alta.

*******************************************************************
        DEF   CSAVE,CLOAD,BLOAD
VMBW    EQU   >2024    :
VMBR    EQU   >202C    :  -> EQUATES per Extended Basic.
VSBR    EQU   >2028    :
ERR     EQU   >2034    :

        AORG  >FDB8          -> Inizio caricamento di questo programma!
INPROG  EQU   >24F0          -> Salva qui' l'indirizzo di inizio programma.
ENPROG  EQU   >24F2          -> Salva qui' l'indirizzo di fine programma.
        B     @CSAVE         -> BRANCH per l'ingresso a CSAVE.
        B     @CLOAD         -> BRANCH per l'ingresso a CLOAD.
        B     @BLOAD         -> BRANCH per l'ingresso a BLOAD.
MYWORK  BSS   >20            -> 32 bytes per il workspace.
SAVE    BYTE  >06            -> Op-code per PAB in VDP RAM per una SAVE.
LOAD    BYTE  >05            -> Op-code per PAB in VDP RAM per una LOAD.
CS1PAB  DATA >0600,>1000,>0000,>2000,>6003  -> PAB per VDP RAM.
        TEXT  'CS1.'         -> Nome del mezzo da azionare (DEVICE NAME).
BLOAD   LWPI  MYWORK         -> BLOAD e' l'ingresso per program files senza DEF,
*                               tipo 5 RUN PROGRAM FILES dell'assembler.
        SETO  R12            -> Identifier di BLOAD e' R12 diverso da zero.
        CLR   R14
        JMP   AFCLO
CLOAD   LWPI  MYWORK         -> CLOAD e' lingresso per caricare programmi che
        CLR   R12               contengono anche la DEF table, tipo XASSEMBLE.
AFCLO   MOVB  @LOAD,@CS1PAB  -> Sistema la copia del  PAB in CPU RAM per una LOAD.
        JMP   NOSAV
```

Questo perché CSALO «DEVE» stare il più possibile vicino a >FFFF: la routine «DEVE» iniziare a >FFD8, pena il non funzionamento del tutto.
Una volta terminato, CSALO permetterà il salvataggio su cassetta dei programmi in L.M. generati da XASSEMBLE, unitamente alla DEF TABLE creata, il LOAD successivo degli stessi, aggiungendo alla DEF TABLE presente le nuove DEF, e terza opzione, il LOAD di program file

```
100 ! Micro e Personal Compu
ter

110 ! SPOLIN : SPOsta
      riferimento LINee di
      un programma Ext.Basic
      con Memory Expansion

120 ! E' da usare per
      costruire XASSEMBLE.

130 ! San Donato MIlanese
      Dicembre 1984,
      Paolo Bagnaresi.

140 CALL LOAD(10054,2,224,39
,38,192,32,131,48,193,32,131
,50,6,4,192,68,208,177,208,2
09,5,192)

150 CALL LOAD(10076,220,2,22
0,3,5,192,129,0,22,251,2,224
,131,224,120,32,131,124,131,
124,4,96)

160 CALL LOAD(10098,0,112,2,
224,39,38,2,0,36,244,2,1,253
,136,2,2,2,48,204,112,6,66)

170 CALL LOAD(10120,22,253,1
6,237)

180 CALL LOAD(16368,82,73,76
,79,67,32,39,116)

190 CALL LOAD(16376,83,80,79
,76,73,78,39,70)

200 CALL LOAD(8194,39,140,63
,216)

210 CALL CLEAR

220 FOR LOC=10054 TO 10123 :
: CALL PEEK(LOC,A):: B=B+A :
: NEXT LOC

230 IF B<>6827 THEN CALL SOU
ND(150,220,0):: PRINT "ERROR
E NEI VALORI DELLE CALL LOAD
, DA LINEA 140 A LINEA 170"
ELSE CALL SOUND(150,1397,0):
: PRINT "OK DA LINEA 140 A L
INEA 170"

240 ! CALL LINK("RILOC")

250 ! CALL LINK("SPOLIN")
```

lunghi fino a quasi 32 Kbytes, sempre da cassetta, se divisi in più spezzoni da 8 Kbytes max. ciascuno. L'ultima opzione può essere utile per caricare giochi in assembler su cassetta. Ce ne sono degli ottimi in giro, generalmente su dischetto e in formato per modulo EDITOR/ASSEMBLER. Non è tuttavia difficile la trasformazione degli stessi per Extended BASIC.
L'altra sera, ad esempio, ho trascorso un'oretta piacevole in una U.S. OPEN, cioè con «TENNIS», il favoloso gioco in L.M. della Nicesoft, che ho caricato da cassetta, in Extended BASIC, con la stessa routine che viene proposta a voi in questo numero. Niente sistema a dischi!

STRUTTURA DI CSALO (SAVE E LOAD)

CSALO viene generato da tre programmi distinti:
1) LINEA25000 (Un programma BAISC fittizio, serve da contenitore)
2) SPOLINEXT (SPOstatore di riferimenti LINee)
3) CSAVEXT (SAVE e LOAD vero e proprio)
LINEA25000 è il programma BASIC «dummy», fittizio, che verrà trasformato da SPOLINEXT.
SPOLINEXT modifica ogni programma BASIC presente in memoria: il risultato è un programma BASIC in cui ogni linea di programma conterrà come istruzioni ciò che era contenuto nella linea di programma a numero più basso. Tutta l'a-

```
25000 !25000
25001 !                        A
                               B
                               C
                               D
                    fin qui-->!
25002 !                        A
                               B
                               C
                               D
                    fin qui-->!
25003 !                        A
                               B
                               C
                               D
                    fin qui-->!
25004 !                        A
                               B
                               C
                               D
                    fin qui-->!
25005 !                        A
              ultima riga: fin
 qui-->U
```

```
NOPRES LI   R0,>2B00      -> Segnala "ERROR NO PROGRAM PRESENT"
       BLWP @ERR
CSAVE  MOVB @SAVE,@CS1PAB  -> Sistema la copia del PAB in CPU RAM per una SAVE.
       LWPI MYWORK
       CLR  R3            -> R3 = zero se non c'e' altro segmento
       MOV  @INPROG,R5    -> Ottiene in R5 l'indirizzo
*                            di inizio caricamento
       JEQ  NOPRES
       MOV  @ENPROG,R4     -> @>2002 c'e' primo indirizzo libero Low Memory
*                             Lo si mette in R4
       JEQ  NOPRES
       S    R5,R4         -> Si ottiene in R4 la lunghezza del program
       JEQ  NOPRES
       LI   R0,>1000      -> Il Buffer e' a >1000 VDP RAM
       LI   R1,MYWORK+6   -> R1 punta cosi' ai 6 bytes da trasferire
       LI   R2,6          -> R2 contiene il numero dei bytes da passare
*                            ora in VDP RAM.
       BLWP @VMBW
       MOV  R5,R1         -> R1 inizio programma in CPU RAM
       MOV  R4,R2         -> R2 numero di bytes del programma in CPU RAM
*                            che unizia dove specificato da R1
       LI   R0,>1006      -> Il programma sara' parcheggiato a partire da
*                            >1006 VDP RAM
       BLWP @VMBW
       A    R2,R0         -> R0 punta cosi' alla prima locazione libera in
*                            VDP RAM, dopo il programma
       MOV  @>2004,R6     -> @>2004 c'e' il puntatore all'inizio della DEF/REF
*                            table. Lo si copia in R6.
       LI   R7,>4000      -> R7 contiene ora la fine della DEF/REF table
       S    R6,R7         -> Ottiene in R7 la lunghezza della DEF/REF table.
       LI   R1,MYWORK+14  -> R1 punta ora a R7
       LI   R2,2
       BLWP @VMBW         -> Passa il numero delle DEFs nel Buffer VDP RAM
       INCT R0            -> R0 punta ora alla locazione successiva in Buffer VDP
       MOV  R6,R1         -> R1 punta a inizio DEF table in CPU RAM
       MOV  R7,R2         -> R2 contiene la lunghezza della DEF TABLE da passare
*                            in VDP RAM.
       BLWP @VMBW
       A    R2,R4         -> R4 contiene ora BYTES PROGRAMMA + BYTES DEF/REF TAB.
       AI   R4,8          -> Aggiunge alla conta i seguenti word in VDP RAM :
*                            >1000, >1002, >1004, >XXXX+2 (vedi sopra a "FORMATO
*                            DEL PROGRAM").
       MOV  R4,@CS1PAB+6  -> Aggiorna la copia in CPU RAM del PAB.
NOSAV  LI   R0,>0F80
       LI   R1,CS1PAB
       LI   R2,14
       BLWP @VMBW         -> Forma il PAB in VDP RAM.
BSEC   LI   R1,>0F8D      ->  SAVE SU CASSETTA : Vedi Man. Ed/As. pag. 253
       MOV  R1,@>8356     ->  Cosi' "POINTER" punta al primo carattere dopo
*                             il nome nel PAB.
       LI   R1,>0003      ->  Sono 3 i caratteri di "CS1"
       MOV  R1,@>8354         e cio' va' a >8354
       LI   R1,>0800      ->  Occorre anche porre >08 a >836D per segnalare
       MOVB R1,@>836D         una DSR CALL.
       LI   R0,>0F8A      ->  Scrive "CS1" sul FAC ( Floating Point Accum.)
```

```
        LI   R1,>834A           che si trova a >834A
        LI   R2,>0003
        BLWP @VMBR
        CLR  @>83D0       -> Occorre anche porre a zero il word a >83D0
        MOVB @>83D0,@>837C -> Occorre settare a zero lo STATUS Byte.
        BLWP @GPLLNK      -> Esegue il "SAVE" su cassetta.
        DATA >003D
*                         -> Va' a vedere se ci sono stati DSR errors.
        LI   R0,>0F81        Leggi il secondo byte PAB in VDP RAM
        BLWP @VSBR
        SRL  R1,13        -> Isola i 3 bytes di sinistra e spostali a destra.
        MOV  R1,R1        -> Vedi se sono zero.
        JEQ  VEDI         -> Si? Allora e' OK.
        LI   R0,>2400     -> No? Allora rendili Most Sig. Byte
        BLWP @ERR         -> Vai a scrivero sullo schermo.
VEDI    CB   @LOAD,@CS1PAB  -> Se l'op-code per una LOAD non e' nella copia CPU
        JNE  RETURN           RAM del PAB, allora siamo in una SAVE.
        LI   R0,>1000     -> Indirizzo VDP RAM del BUFFER.
        LI   R1,MYWORK+6  -> R1 punta a R3 del workspace in uso (MYWORK).
        LI   R2,6         -> Sono 3 i words che servono per ident. un programma.
        BLWP @VMBR
        MOV  R3,R13       -> Salva Identifier NEXT PROGRAM.
        LI   R0,>1006     -> Punta all'inizio del programma in VDP RAM.
        MOV  R4,R2        -> R4 = numero bytes del programma.
        MOV  R5,R1        -> R5 = inizio di caricamento in CPU RAM del programma.
        BLWP @VMBR
        ABS  R12          -> Se R12 = zero  non siamo in BLOAD.
        JEQ  NOBLO
        MOV  R14,R14      -> Se R14 = zero e' il primo program a essere caricato.
        JNE  SECOND
        MOV  R1,R14       -> Salva in R14 l'indirizzo di inizio, una sola volta.
SECOND  ABS  R13          -> Se R13 = zero e' l'ultimo program a essere caricato.
        JNE  BSEC
        MOV  R14,@>83E0   -> Mette in R0 di GPL Workspace l'indirizzo di inizio.
        LWPI >83E0        -> L'esecuzione dei programmi inizia sempre dal GPL
*                            Workspace (>83E0).
        MOV  R11,@SAVRET  -> Salva indirizzo di ritorno del GPL Workspace.
        BL   *R0          -> Esegue i programmi caricati.
        LWPI MYWORK       -> Ripristina il worskspace in uso precentemente.
        MOV  @SAVRET,@>83F6 -> Ripristina R11 di GPL Workspace.
        JMP  RETURN
SAVRET  DATA >0000        -> Salva qui' l'indirizzo di ritorno del GPL.
NOBLO   MOV  R5,@>24F0    -> >24F0 : Locaz. convenzionale (mia) per inizio
*                                    programma.
        MOV  R5,@>24F2
        A    R4,@>24F2    -> >24F2 : Locaz. convenzionale (mia) per fine
*                                    programma.
        C    R5,@>2002    -> Il puntatore @>2002 (prima locaz. di memoria
        JNE  AORGED          libera), viene aggiornato solo se coincide con
*                            indirizzo di caricamento.
        A    R4,@>2002    -> Aggiorna puntatore prima locazione di memoria
*                            libera.
*
```

rea di memoria, impegnata precedentemente dalle istruzioni BASIC appartenenti alle linee successive alla prima, si rende libera e atta a contenere un programma in L.M..
Il programma ottenuto non può essere modificato, pena la distruzione totale dello stesso (collasso generale). Può essere però salvato e richiamato, da disco e da cassetta. Può essere addizionato anche di nuove linee, sia più alte che più basse, o anche intermedie a quelle già esistenti. Non si può effettuare il MERGE di questo genere di programmi, ma essi possono ESSERE addizionati di un programma esterno in formato MERGE. La porzione BASIC aggiunta successivamente funzionerà in maniera completamente normale: potrà a sua volta essere variata, annullata, si potrà usare il REDO per la modifica dei numeri di linea, purché ci si mantenga fuori dell'area originaria, quella sottoposta a SPOLINEXT. il RESequence funziona regolarmente, etc., etc.. Unica cosa, non modificate MAI le linee modificate da SPOLINEXT.
CSAVEXT è la routine di CSALO ottenuta con CALL LOAD. Viene fornita anche la sorgente, che si noterà, contiene una AORG a >FFD8, mentre le CALL LOAD si riferiscono alla zona di bassa memoria. Ci penserà successivamente un rilocatore, contenuto in SPOLINEXT, a rilocare CSAVEXT da bassa ad alta memoria.

COME SI PROCEDE

Non è facile, ogni errore di battitura si tradurrà in una mancanza di funzionamento finale. Se avete amici, datevi il turno, soprattutto per le CALL LOAD.
1) Innanzi tutto battere LINEA25000. È importante riempire le linee in modo che, alla fine, il SIZE sia di 23852 bytes of PROGRAM SPACE FREE. Se lo SPACE FREE sarà di meno, non funzionerà il programma, se sarà di più, potreste non essere poi in grado di salvare il tutto su cassetta, perché la lunghezza totale di XASSEMBLE supererà la lunghezza massima consentita per una SAVE su cassetta di un programma BASIC con espansione di memoria inserita. Nel listato, la freccia che punta alla «!» di fine linea dovrebbe aiutare in fase di battitura e, comunque, la fine linea coincide, all'infuori della prima e dell'ultima linea, con l'arresto forzato del cursore (BEEP). Terminato di battere, effettuare un SAVE CS1 (DSK1) di LINEA25000.
2) Fate un NEW, e battete SPOLINEXT. Finito di battere, fate un SAVE. Poi una CALL INIT, solo questa volta, poi mai più fino a spegnimento del computer. Poi

RUN: se non otterrete l'OK, correggete le linee segnate. Caricate ora (OLD) LINEA25000, fate una CALL LINK («SPOLIN»), e poi LIST. Se SPOLIN era battuto correttamente il vostro listato sarà così: 25000 !25000, 25001 !25000, etc, cioè tutte le linee punteranno al contenuto di linea 25000, altrimenti riguardate con più attenzione SPOLINEXT: potreste aver rovesciato qualche dato. Fate una CALL LINK («RILOC»): il cursore deve tornare subito, e apparentemente nulla sarà mutato.

3) Fate un NEW, e battete CSAVEXT. Fate poi un SAVE, quindi un RUN. Se avrete l'OK è molto probabile che sia effettivamente tutto giusto. Gli unici errori possibili sarebbero quelli di inversione dei valori, o, più raramente, una compensazione interna fra gli errori.

4) Ora eseguite una OLD per LINEA25000. Eseguite una CALL LINK («RILOC»). Alla riapparizione immediata del cursore avrete completato la preparazione di CSALO. Salvatelo su cassetta.

5) Per controllo eseguite una CALL LINK («CSAVE»). Dovreste ottenere un «WARNING NO PROGRAM PRESENT». Infatti per CSAVEXT un programma in L.M. inizia, e termina, alle locazioni indicate, rispettivamente, dal contenuto di >24FO, e >24F2 (decimali 9456 e 9458). Siccome normalmente queste due locazioni sono zero, si ottiene una segnalazione di errore.

6) Altro controllo: CALL LINK («CLOAD»). Dovreste ottenere l'avvio della routine di salvataggio su cassetta: «REWIND CASSETTE TAPE THEN PRESS ENTER». Se ciò non accade riguardarsi tutto CSAVEXT e anche SPOLINEXT: il rilocatore è in SPOLINEXT. Se invece và tutto bene premete «E», perché al momento molto probabilmente non avrete nulla da caricare da cassetta come program file: un programma BASIC non và bene, occorre o un program generato da CSAVE, o un program generato dalla routine «SAVE» del modulo EDITOR/ASSEMBLER.

Chi ha l'EDITOR/ASSEMBLER si comporti così:

1) Generare sia SPOLIN che CSAVE, con il listato riportato per l'EDITOR, e assemblare in formato non compresso.

2) Comporre, da Ext. BASIC, solo LINEA25000.

3) Caricare SPOLIN <CALL INIT::CALL LOAD («DSK1.SPOLIN»)::CALL LINK («SPOLIN»)>.

4) Caricare, solo ora, non prima, CSAVE <CALL LOAD («DSK1.CSAVE»)>. CSAVE andrà a piazzarsi a iniziare da >FFD8. Non fate CALL LINK («RILOC»),

```
AORGED A    R2,R0          -> R0 punta ora al byte lunghezza DEF table in VDP RAM
       LI   R1,MYWORK+4    -> R1 punta a R2
       LI   R2,2
       BLWP @VMBR          -> Ottiene in R2 lunghezza DEF table.
       MOV  @>2004,R1      -> @>2004 puntatore all'inizio vecchia DEF table :
*                             viene copiato in R1
       S    R2,R1          -> R1 punta cosi' tanti bytes piu' in basso, quanto e'
*                             e' lunga l'aggiunta alla DEF table.
       INCT R0             -> R0 punta ora a inizio copia DEF table in VDP RAM
       BLWP @VMBR          -> Ottiene aggiunta alla DEF table in CPU RAM
       MOV  R1,@>2004      -> Aggiorna il puntatore alla DEF table.
RETURN CLR  R0             ->  Ritorna all'EXTENDED BASIC
       MOVB R0,@>837C
       LWPI >83E0
       B    @>0070
GPLLNK DATA WORKSP,COUNTE    -> Vettori per la BLWP @GPLLNK.
WORKSP BSS  >20              -> Workspace di lavoro.
RETDAT DATA RETADD           -> Vettore per il ritorno dalla GPL routine al nostro
*                               programma.
COUNTE MOVB @>9802,R1        -> Legge a che indirizzo sta puntando ora il
*                               puntatore alla GROM.
       SWPB R1
       MOVB @>9802,R1        -> Sono due i bytes che compongono un indirizzo in
*                               in GROM, cosi' come in CPU RAM e VDP RAM.
       SWPB R1               -> Il primo a essere traferito e' il Most Significant
       DEC  R1
       MOV  R1,R5            -> Salva in R5 il vecchio indirizzo.
       DECT R1               -> In R1 c'e' l'indirizzo GROM per una XMLLNK.
       MOVB @>8373,R2        -> Copia il puntatore al RETURN STACK.
       SRL  R2,8             -> Rendilo Least Significant Byte.
       AI   R2,>8300
       INCT R2               -> R2 punta cosi' ad una nuova routine nel SUBROUTINE
       MOVB R1,*R2           -> Il primo byte della routine e' cosi' sistemato
       SWPB R1               -> Si appresta a passare il secondo byte.
       MOVB R1,@>0001(R2)    -> Passa ora il secondo byte dell'indirizzo GROM.
       SWPB R2               -> Risistema i Bytes all'interno di R2.
       MOVB R2,@>8373        -> Risistema il SUBROUTINE STACK POINTER.
       MOVB *R14+,@>9C02     -> Traferisce il DATA seguente la BLWP @GPLLNK nel
       MOVB *R14+,@>9C02        puntatore all'indirizzo GROM: >9C02 e' il GRMWA.
       MOV  @>2000,R4        -> Salva in R4 il vettore di entrata per CALL LINK.
       MOV  @RETDAT,@>2000   -> Inserisce il vettore al nostro ritorno.
       LWPI >83E0            -> GPL routine parte sempre da GPL Workspace : >83E0
       B    @>0070           -> Si allaccia all'interprete GPL.
RETADD LWPI WORKSP           -> Se tutto e' ok e' qui che ritorna alla nostra
*                               GPLLNK routine.
       MOV  R4,@>2000        -> Restore il vecchio vettore per il controllo
*                               DEF-CALL LINK.
       MOVB R5,@>9C02        -> Ripristina il vecchio indirizzo GROM.
       SWPB R5               -> Si prepara a passare anche il byte di destra.
       MOVB R5,@>9C02
       RTWP                  -> Fine della GPLLNK e ritorno, dalla BLWP, al
*                               segmento che ha chiamato la BLWP @GPLLNK.
       END
```

perché non c'è proprio nulla in bassa memoria da rilocare.
5) Avrete così ottenuto CSALO. Salvatelo su disco.
Chi ha la MINI-MEMORY non potrà giovarsene.

COME SI UTILIZZA CSALO

CSALO, oltre a essere inserito in XASSEMBLE, può essere usato anche da solo.
All'inizio del programma, vedi listato assembler, ci sono tre BRANCH: ognuna porta ad una routine diversa.
>FD88 (bytes dec. 253,136) -> porta a CSAVE
>FD8C (bytes dec. 253,140) -> porta a CLOAD
>FD8C (bytes dec. 253,144) -> porta a BLOAD
CSAVE stà per «Cassette SAVE», CLOAD per «Cassette LOAD», BLOAD per «Branch all'indirizzo di primo caricamento dopo LOAD».

Appena formato, CSALO ha anche le DEF pronte, ma una volta che spegnete il computer le DEF spariscono, e non riappaiono più, neppure dopo una OLD di CSALO. E allora? È semplice, sfruttiamo il puntatore a >2000, risparmiando così memoria. Evitiamo anche, così, la duplicazione delle DEF CSAVE, CLOAD, BLOAD, che avremmo dopo l'utilizzo della routine CSALO da parte di XASSEMBLE. Abbiamo parlato del puntatore a >2000, nel numero di novembre. Lo

```
100 ! Micro & Personal Compu
ter

110 ! CSAVEXT, ad AORG >FD88
      rilocato in bassa
      memoria,
      da >24F4 a >2724

120 ! Deve essere rilocato
      a >FD88 con SPOLINEXT
      (vedi) prima dell'uso.

130 ! Da usare per routines
      "SAVE" o  "LOAD" con
      cassetta, o da solo o
      per XASSEMBLE.

140 ! San Donato Milanese
      Dicembre 1984
      Paolo Bagnaresi

150 CALL LOAD(16344,66,76,79
,65,68,32,253,196)

160 CALL LOAD(16352,67,76,79
,65,68,32,253,206)

170 CALL LOAD(16360,67,83,65
,86,69,32,253,228)

180 CALL LOAD(8194,39,140,63
,216)

190 CALL LOAD(9460,4,96,253,
228,4,96,253,206,4,96,253,19
6,0,0,0,0,0,0,0,0,0)

200 CALL LOAD(9482,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0)

210 CALL LOAD(9504,6,5,6,0,1
6,0,0,0,32,0,96,3,67,83,49,4
6,2,224,253,148,7,12)

220 CALL LOAD(9526,4,206,16,
3,2,224,253,148,4,204,216,32
,253,181,253,182,16,54,2,0,4
3,0)

230 CALL LOAD(9548,4,32,32,5
2,216,32,253,180,253,182,2,2
24,253,148,4,195,193,96,36,2
40,19,243)

240 CALL LOAD(9570,193,32,36
,242,19,240,97,5,19,238,2,0,
16,0,2,1,253,154,2,2,0,6)

250 CALL LOAD(9592,4,32,32,3
6,192,69,192,132,2,0,16,6,4,
32,32,36,160,2,193,160,32,4)

260 CALL LOAD(9614,2,7,64,0,
97,198,2,1,253,162,2,2,0,2,4
,32,32,36,5,192,192,70)

270 CALL LOAD(9636,192,135,4
,32,32,36,161,2,2,36,0,8,200
,4,253,188,2,0,15,128,2,1)

280 CALL LOAD(9658,253,182,2
,2,0,14,4,32,32,36,2,1,15,14
1,200,1,131,86,2,1,0,3)

290 CALL LOAD(9680,200,1,131
,84,2,1,8,0,216,1,131,109,2,
0,15,138,2,1,131,74,2,2)

300 CALL LOAD(9702,0,3,4,32,
32,44,4,224,131,208,216,32,1
31,208,131,124,4,32,255,56,0
,61)

310 CALL LOAD(9724,2,0,15,12
9,4,32,32,40,9,209,192,65,19
,4,2,0,36,0,4,32,32,52)

320 CALL LOAD(9746,152,32,25
3,181,253,182,22,62,2,0,16,0
,2,1,253,154,2,2,0,6,4,32)

330 CALL LOAD(9768,32,44,195
,67,2,0,16,6,192,132,192,69,
4,32,32,44,7,76,19,19,195,14
2)

340 CALL LOAD(9790,22,1,195,
129,7,77,22,191,200,14,131,2
24,2,224,131,224,200,11,254,
244,6,144)

350 CALL LOAD(9812,2,224,253
,148,200,32,254,244,131,246,
16,27,0,0,200,5,36,240,200,5
,36,242)

360 CALL LOAD(9834,168,4,36,
242,136,5,32,2,22,2,168,4,32
,2,160,2,2,1,253,152,2,2)

370 CALL LOAD(9856,0,2,4,32,
32,44,192,96,32,4,96,66,5,19
2,4,32,32,44,200,1,32,4)

380 CALL LOAD(9878,4,192,216
,0,131,124,2,224,131,224,4,9
6,0,112,255,60,255,94,0,0,0,
0)

390 CALL LOAD(9900,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0)

400 CALL LOAD(9922,0,0,0,0
,0,255,164,208,96,152,2,6,
3,208,96,152,2,6,193,6,1)

410 CALL LOAD(9944,193,65,
65,208,160,131,115,9,130,2
4,131,0,5,194,212,129,6,19
216,129)

420 CALL LOAD(9966,0,1,6,1
,216,2,131,115,216,62,156,
216,62,156,2,193,32,32,0,2
,32)

430 CALL LOAD(9988,255,92,
,0,2,224,131,224,4,96,0,11
2,224,255,60,200,4,32,0,21
5)

440 CALL LOAD(10010,156,2,
197,216,5,156,2,3,128,0,0)

450 CALL CLEAR

460 FOR LOC=9460 TO 10019
 CALL PEEK(LOC,A):: B=B+A
 NEXT LOC

470 IF B<>41238 THEN CALL
UND(150,220,0):: PRINT "ER
RE NEI VALORI DELLE CALL L
D, DA LINEA 190 A LINEA 44
 ELSE CALL SOUND(150,1397,
:: PRINT "OK DA LINEA 190
LINEA 440"

480 ! CALL LINK("BLOAD")

490 ! CALL LINK("CSAVE")

500 ! CALL LINK("CLOAD")
```

utilizziamo anche ora per collegarci direttamente alla porzione di programma in L.M. che ci interessa.
Dopo la OLD di CSALO, eseguire una CALL INIT e poi:
— per CSAVE: CALL LOAD (8192,253,136)::CALL LINK («CSAVE»)::CALL LOAD (8192,32,90)
— per CLOAD: CALL LOAD (8192,253,140)::CALL LINK («CLOAD»)::CALL LOAD (8192,32,90)
— per BLOAD: CALL LOAD (8192,253,144)::CALL LINK («BLOAD»)::CALL LOAD (8192,32,90)
Notare che, dopo la prima CALL LOAD, il nome della CALL LINK può essere a piacere, il BRANCH avverrà ugualmente all'indirizzo dato dalla CALL LOAD. L'ultima CALL LOAD serve per ripristinare il puntatore in modo che segni >2050, altrimenti qualsiasi altra CALL LINK, per altri programmi in L.M. eventualmente presenti in memoria, porterebbe inevitabilmente sempre all'ultima routine richiesta con questo tipo particolare di ingresso ad un programma in L.M..
CSAVE può essere utilizzata solo dopo la sistemazione dell'indirizzo di inizio di caricamento e dell'indirizzo di fine caricamento, altrimenti otterrete un «WARNING NO PROGRAM PRESENT». Le locazioni sono, come detto sopra, >24FO (dec. 9456), >24F2 (dec. 9458). È un po' complesso: se il programma da salvare inizia a >24F4, e termina a >3022, procedere così: scomporre in bytes gli indirizzi, trasformarli in decimale, ed effettuare CALL LOAD per i valori ottenuti. Nell'esempio >24 = 36, >F4 = 244, >30 = 48, >22 = 34. Eseguire quindi: CALL LOAD (9456,36,244,48,34).
CSALO può essere utilizzato in qualsiasi programma Ext.BASIC di vostra creazione, per richiamare routine in L.M. che creerete con XASSEMBLE, a patto che sia la prima porzione del programma che ideerete.
Buon lavoro.

Paolo Bagnaresi

LISTATI DEI PROGRAMMI

Da questo numero in poi i listati BASIC ed EXTENDED BASIC appariranno stampati sulla nostra rivista così come essi vi appariranno sul video dopo essere stati digitati. Ciò vi faciliterà il compito di debugging (ricerca degli errori di battitura): basterà osservare un disallineamento tra il vostro listato e quello originale della rivista per capire che avete aggiunto o tolto qualche carattere.
I programmi saranno, nei limiti del possibile, tutti RESequenziati da linea 100, con un incremento di 10 fra linea e linea.
Ciò faciliterà la battitura: si imposta NUM e si procede senza dover aggiungere, per esempio, la riga 277 o la riga 903. Inoltre, questo metodo eliminerà la possibilità di dubbi (ex: ma questo qui che accidente è, un DATA o un numero di linea?). Coloro che intendono inviare programmi BASIC per pubblicazione, eseguano un RES prima di accingersi a descrivere l'uso dei blocchi del programma, e poi descrivano il loro programma in base ai nuovi numeri di linea ottenuti.

RISPOSTE IN BREVE

— Alcuni lettori hanno avuto difficoltà ad adattare i programmi Assembler, pubblicati nei numeri scorsi, alla Mini Memory. Purtroppo tali programmi, così come sono stati pubblicati, non possono venire accettati dall'assemblatore line-by-line. Ci ripromettiamo, in uno dei prossimi numeri, di tornare più diffusamente sull'argomento. — Alcuni lettori chiedono informazioni riguardanti libri che trattino l'Assembler del TI-99. Le pubblicazioni reperibili purtroppo sono poche, ma fortunatamente ben scritte. Eccone comunque un elenco: 1) Denise Amrouche-Roger Didi: Initiation au Langage Assembleur du Texas Instruments TI-99/4A. Ed. SHIFT-27, Rue du General Foy-75008 PARIS (FRANCIA). 2) M.S. Morley: Fundamentals of TI-99/4A Assembly language - TAB Books Inc. Blue Ridge Summit, PA 17214-USA. 3) Molesworth: Introduction to Assembly Language - Editore non noto. Tale volume è reperibile presso la NOVEL 81 - V. Crispi, 57 - Roma - Tel. 06/6783424. 4) Beginning Assembly Language for the TI Home Computer - D&D Publishing CO. - 3177 Bellevue - Toledo - OHIO 43606-USA. Di quest'ultimo libro abbiamo notizie molto scarse e non lo conosciamo direttamente, pertanto, se qualche lettore avesse notizie più dettagliate, saremmo ben lieti di farle conoscere a tutta la comunità del TI-99. Presso la Novel 81, Via F. Crispi 57, Roma, è disponibile per il TI-99 Hardware e Software importato direttamente dagli USA. Chi ha problemi di reperibilità del modulo Extended Basic o espansione di memoria laterale a 32 K può contattare la NEWSOFT, Via Jacini 4, Milano. — Alcuni lettori vogliono sapere più in dettaglio sotto quale forma si possono mandare programmi per l'eventuale pubblicazione sulla rivista. Vediamo di riepilogare le condizioni a cui si devono attenere i lettori: 1) Inviare il programma su cassetta o floppy corredato da un commento dettagliato. Ricordarsi di inserire anche nel programma eventuali commenti. Il listato non è indispensabile, meglio se c'è. 2) I programmi pubblicati verranno rimunerati con una cifra variabile a partire da L. 40.000 fino a L. 100.000. 3) Il materiale, inviato per la pubblicazione, non viene restituito. 4) Programmi molto lunghi e particolarmente interessanti che, seppur meritevoli, per esigenze di spazio non possono venire pubblicati, potranno comunque venire recensiti dalla redazione.

colloquio. In questa direzione quindi vanno concentrati gli sforzi, va discusso il problema, vanno cercate soluzioni: quando si comincerà ad operare così, vedrete che anche il 1650 arriverà in Italia.

D Distinti Signori, un paio di anni fa ho comperato un computer, non certo un giocattolo ma il più perfezionato fra quelli che non hanno floppy o monitor incorporato. È dopo un certo tempo, ho cominciato ad avere l'impressione che qualcosa non andava. Non nel mio apparecchio, ma piuttosto nel mondo dei micro e dei personal in genere. Ho apprezzato sempre meno le ben conosciute limitazioni, per esempio dovere battere un programma lungo, complicato e puntiglioso (e non ho nemmeno Pascal!) per qualunque cosa, anche le funzioni più semplici. Mi è sembrato avere in mano, se non un dinosauro, almeno una specie di auto, modello 1905: un'apparecchio limitato, complicato, esigente e «nudo»: a quei tempi si comperava spesso un telaio nudo, che bisognava poi fare carrozzare da un vero carrozziere.
Ahimé, se nelle auto 1900 la carrozzeria era fatta una volta per sempre (o quasi), nei micro e personal bisogna in un certo modo rifarla ogni volta che vuole fare qualunque cosa... e ho cominciato a odiare l'idea di dovere dipendere da software o programmi difficili e complicati per ogni cosa che intendo fare.
Ho abitato molti anni nel Nord Europa, dove la gente magari piagnucola davanti alle difficoltà, ma poi con lodevole testardaggine si rimbocca le maniche e tenta almeno qualcosa, e mi sono messo al lavoro per concepire qualcosa *a misura di utente*, cioè ultra-facile da usare da chiunque, soprattutto da chi ha ben altri grattacapi per mettersi a ragionare in termini logico-matematici o di BASIC. Così cominciai a mettere a punto ISY (o SIMPLEX, © 1983).
Oggi, non solo non rinnego ISY ma peggio (perseverare diabolicum!), alcuni strani dubbi mi passano per la testa: e se l'industria dei piccoli computer fosse condannata a sparire quasi completamente entro pochissimi anni? E se il personal passasse di moda, esattamente come la CB, le calcolatrici tascabili e altre mode? Chi mi assicura che la gente non si stuferà entro 2-3 anni dei soliti giochetti sempre uguali, dei soliti alieni da disintegrare, dei soliti labirinti? Resisterà il micro alla prossima ondata di super-gadget, di TV a schermo piatto, con alta definizione, stampante, Videotext e chissà quanto e che altro?
Insomma, penso che il computer di gestione resterà, ma il computer fasullo e per giochetti è destinato a una fine rapida, o a una svalutazione al rango di giocattolo made in Taiwan, molto prima di quanto molti credono, e questo solo perché quando il prodotto «tirava», nessuno si è preoccupato di farlo meglio, o almeno semplice per l'utente. Forse tutti hanno sbagliato a credere in un «futuro pieno di computer», così come altri hanno sbagliato 15-20 anni fa a immaginare che oggi, come perforatori e perforatrici di schede e nastri da grossi computer di gestione, avrebbero dovuto lavorare centinaia di migliaia di persone.
Forse i più intraprendenti dei vostri lettori, coloro che pensano sistemarsi un giorno per conto loro, saranno interessati in ISY; ormai, sistemi con Z80, BASIC e 64K se ne faranno anche perfino in Mongolia, e ci vuole ben altro per reggere la concorrenza e stare al passo con i tempi. ISY significa un linguaggio diretto, o un'interprete di cui molte istruzioni sono spesso microprogrammi o spezzoni di programma in ROM. Ciò potrebbe comportare diversi «banchi» di ROM per un totale di magari 200 K (e quasi ci siamo alle memorie di 256 K) e un funzionamento forse più lento, almeno all'inizio. Penso che *non serve a niente avere computer che fanno milioni di operazioni al secondo, se poi ci vogliono milioni di secondi* (= settimane, debugging compreso) *per tirarne fuori un buon programma.* Con ISY o qualunque roba simile, buona parte del software destinato al pubblico in genere diventa inutile; perché chiunque può rifarlo a casa e anche meglio, dopo aver letto un manuale non più complicato di una rivista a fumetti. Per certe applicazioni il BASIC resta la migliore soluzione, per altre è il contrario, ma i sistemi non sono comparabili né direttamente concorrenti, visto che il BASIC è ben oltre la portata e la pazienza della maggioranza degli utenti di computer.
Allego alcuni esempi di istruzioni in ISY, che sottometto al vostro giudizio imparziale e indipendente, e vi chiedo inoltre di fare sapere ai vostri lettori l'esistenza o le possibilità di questo tipo di linguaggio. Dopodiché, se qualcuno è interessato, lo faccia sapere, se no tanti saluti e amici come prima!
Distinti saluti

ROBERT LOMBARDI
ROMA

R *Pubblichiamo questa lettera che tutto sommato contiene delle idee che possono destare interesse e voglia di discutere. Gli esempi inviati a noi sono troppo lunghi per poter essere pubblicati, ma credo che il sig. Lombardi non abbia problemi a chiarirsi con chi è interessato ad approfondire la questione; l'idea di fondo del suo linguaggio è buona: non ci sentiamo di aggiungere altro, vista la grande quantità di linguaggi di programmazione, più o meno diffusi, in circolazione nel mondo.*

Nella rubrica Texas del numero scorso nel montaggio dell'ultima parte di listato riguardante XASSEMBLE si è persa l'ultima colonna del testo, che quindi a partire dall'istruzione 400 del listato a pag. 120 manca dell'ultimo numero in ogni riga. Ripubblichiamo quindi qui sotto la parte incompleta, scusandoci per l'inconveniente.

```
400 CALL LOAD(9922,0,0,0,0,0
,0,255,164,208,96,152,2,6,19
3,208,96,152,2,6,193,6,1)

410 CALL LOAD(9944,193,65,6,
65,208,160,131,115,9,130,2,3
4,131,0,5,194,212,129,6,193,
216,129)

420 CALL LOAD(9966,0,1,6,194
,216,2,131,115,216,62,156,2,
216,62,156,2,193,32,32,0,200
,32)

430 CALL LOAD(9988,255,92,32
,0,2,224,131,224,4,96,0,112,
2,224,255,60,200,4,32,0,216,
5)

440 CALL LOAD(10010,156,2,6,
197,216,5,156,2,3,128,0,0)

450 CALL CLEAR

460 FOR LOC=9460 TO 10019 ::
 CALL PEEK(LOC,A):: B=B+A ::
 NEXT LOC

470 IF B<>41238 THEN CALL SO
UND(150,220,0):: PRINT "ERRO
RE NEI VALORI DELLE CALL LOA
D, DA LINEA 190 A LINEA 440"
 ELSE CALL SOUND(150,1397,0)
:: PRINT "OK DA LINEA 190 A
LINEA 440"

480 ! CALL LINK("BLOAD")

490 ! CALL LINK("CSAVE")

500 ! CALL LINK("CLOAD")
```